

NAME

motors – commands and functions for controlling motors

DESCRIPTION

Motor positions are stored in memory, in the *settings* file and, if the hardware allows, in motor-controller registers. The user has access to the motor positions through the built-in array `A[]`. The number of steps per degree (or millimeter, inch, etc.), direction of motion, rate of rotation, direction of backlash, etc., are defined in a configuration file. Each motor has an associated name and mnemonic, also assigned in the configuration file.

`spec` employs two methods for specifying motor position, dial units and user units. Dial units should be set to correspond to the physical dial settings of the spectrometer. Doing so makes it more likely you can recover motor positions in the event controller and software positions are simultaneously lost. User units are related to dial units by a simple linear relation. You set user angles to nominal values during your lineup procedure.

The *settings* file contains a copy of the most recent value written to the motor controller register, the user-units offset and the software limits for each motor. The *settings* file is read at program start up and on the `reconfig` command. The *settings* file is written each time a motor is moved or when the offset or limits are changed.

Dial and user units are related by the following expressions.

$$\begin{aligned} user &= sign \times dial + offset \\ dial &= controller_register / steps_per_unit \end{aligned}$$

sign and *steps_per_unit* are set in the configuration file.

For example, if *sign* = 1 and *steps_per_unit* = 200, the current position would be

$$user = controller_register / 200 + offset$$

Motor positions displayed by `spec` are generally rounded to the precision allowed by the steps-per-unit parameter. With `spec` release 5.10.01-1, it is now possible to retrieve the “commanded” motor position, which is the position requested with a move command. The commanded position can be retrieved using `read_motors()`.

BUILT-IN COMMANDS, FUNCTIONS AND VARIABLES

`A[]` – An array used to transfer motor positions between the user and the program.

`read_motors(mode[, which])` – Reads the motors and places the motor positions in the `A[]` array with options set by *mode*, as follows:

- Bit 0 (0x01) – If clear, user positions are put into `A[]`. If set, dial positions are put into the motor array.
- Bit 1 (0x02) – If set, forces reading the hardware. For many motor controllers, `spec` doesn’t necessarily read the hardware if the position hasn’t been changed by `spec` since the controller was last read.
- Bit 2 (0x04) – If set, position discrepancies between `spec` and the motor hardware will be silently resolved in favor of the hardware. Otherwise, `spec` will prompt the user as to whether the software or hardware positions should be considered correct.
- Bit 3 (0x08) – If set, the hardware is read, but the contents of the `A[]` array are not modified.
- Bit 4 (0x10) – If set, the “commanded” positions are placed in `A[]`. The hardware is not accessed. If the optional argument *which* is set to a particular motor number or mnemonic, `A[]` is not modified, and the return value will be the commanded position for the specified motor.

The optional argument *which* is currently only used for return of the commanded position.

The following macro definitions are included in the standard set:

```
def getangles 'read_motors(0)'  
def getdials  'read_motors(1)'  
def get_angles 'read_motors(0); user_getangles'
```

The `get_angles` definition includes a call to user-hook macro, `user_getangles`, that can be defined locally. It is recommended to use `get_angles` in local macros.

Note, prior to the introduction of `read_motors()` in `spec` release 4.03.07, `getangles` and `getdials` were built-in commands.

`move_all` – Move all motors to the user positions specified in `A[]`.

The following macro definition is included in the standard macro set:

```
def move_em 'user_premove; move_all; user_postmove'
```

The `move_em` macro includes calls to user-hook macros that can be defined locally. It is recommended that local macros use `move_em` rather than `move_all`.

`move_cnt` – As `move_all`, but gates scalers open during move and does not perform backlash correction. Move velocity is at the base rate.

`motor_mne(i)` – Returns the string mnemonic of motor *i* as given in the configuration file.

`motor_name(i)` – Returns the string name of motor *i* as given in the configuration file.

`motor_num(mne)` – Returns the motor number corresponding to the motor mnemonic *mne*, or -1 if there is no such motor configured.

`motor_par(i, s [, v])` – Returns or sets configuration parameters for motor *i*. Recognized values for the string *s* include:

"step_size" – returns the current step-size parameter. Unfortunately, this command is misleadingly named. The value returned is actually the number of steps per unit (where the unit is generally degrees, millimeters, microns, etc.), *not* the size of a single step. If *v* is given, then the step size is set to that value.

Note, the parameter can be negative, which might be needed to make the dial position agree with the rotation sense of the motor.

Since changing this parameter seriously affects the motor position calculation, as a precaution, the function `spec_par("modify_step_size", 1)` must be entered first to enable step-size modifications using `motor_par()`.

"sign" – returns 1 or -1, indicating the rotation sense of the user angle versus the dial angle, as set in the *config* file (as of `spec` release 5.08.01-5). This is a read-only parameter.

"offset" – returns the value of the "offset" motor parameter, which is the difference between the user and dial motor positions (as of `spec` release 5.08.01-1). This is a read-only parameter.

"acceleration" – returns the value of the current acceleration parameter. The units of acceleration are the time in milliseconds for the motor to accelerate to full speed. If *v* is given, then the acceleration is set to that value.

"base_rate" – returns the current base-rate parameter. The units are steps per second. If *v* is given, then the base rate is set to that value.

- "velocity" – returns the current steady-state velocity parameter. The units are steps per second. If v is given, then the steady-state velocity is set to that value.
- "backlash" – returns the value of the backlash parameter. Its sign and magnitude determine the direction and extent of the motor's backlash correction. If v is given, then the backlash is set to that value. Setting the backlash to zero disables the backlash correction.
- "config_step_size" – Returns the value of the "step_size" parameter as set in the *config* file.
- "config_acceleration" – Returns the value of the "acceleration" parameter as set in the *config* file.
- "config_base_rate" – Returns the value of the "base_rate" parameter as set in the *config* file.
- "config_velocity" – Returns the value of the "velocity" parameter as set in the *config* file.
- "config_backlash" – Returns the value of the "backlash" parameter as set in the *config* file.
- "controller" – returns a string containing the controller name of the specified motor. The controller names are those used in *spec*'s *config* files.
- "unit" – returns the unit number of the specified motor. Each motor controller unit may contain more than one motor channel.
- "channel" – returns the channel number of the specified motor.
- "responsive" – returns a nonzero value if the motor responded to an initial presence test or appears otherwise to be working.
- "active" – returns a nonzero value if the motor is currently moving.
- "high_lim_hit" – returns nonzero if the high limit is active (for most supported motor controllers) (as of *spec* release 5.06.05.7).
- "low_lim_hit" – returns nonzero if the low limit is active (for most supported motor controllers) (as of *spec* release 5.06.05.7).
- "disable" – returns a nonzero value if the motor has been disabled by software. If v is given and is nonzero, then the motor is disabled. If v is given and is zero, the motor becomes no longer disabled. A disabled motor channel will not be accessed by any of *spec*'s commands, and, of course, cannot be moved. Any `cdef()`-defined macros will automatically exclude the portions of the macro keyed to the particular motor when the motor is software disabled.
- "writable" – returns a value indicating the permission status of the motor, as set in the *config* file. If bit 1 is set, the motor can be moved. If bit 2 is set, the limits can be changed. A fully protected motor will return zero. A fully open motor will return 3. (Added in *spec* release 5.08.01-1.) This is a read-only parameter.
- "slop" – returns the value of the slop parameter. If v is given, sets the slop parameter. When this parameter is greater than zero, discrepancies between hardware and software motors positions are silently resolved in favor of the hardware when the number of steps in the discrepancy is less than the value of the slop parameter. If the value is negative, discrepancies less than the absolute value of the parameter are resolved in favor of software by changing the hardware controller position, if possible.
- "home_slew_rate" – returns the value of the home-slew-rate parameter. If v is given, sets the parameter. This parameter is the steady-state velocity used

during a home search. (Only available for selected controllers.)

"home_base_rate" – returns the value of the home-base-rate parameter. If v is given, sets the parameter. This parameter is the base-rate velocity used during a home search. (Only available for selected controllers.)

"home_acceleration" – returns the value of the home-acceleration parameter. If v is given, sets the parameter. This parameter is the acceleration used during a home search. (Only available for selected controllers.)

"dc_dead_band" – returns the value of the dead-band parameter for DC motors. Sets the parameter if v is given.

"dc_settle_time" – returns the value of the settle-time parameter for DC motors. Sets the parameter if v is given.

"dc_gain" – returns the value of the gain parameter for DC motors. Sets the parameter if v is given.

"dc_dynamic_gain" – returns the value of the dynamic-gain parameter for DC motors. Sets the parameter if v is given.

"dc_damping_constant" – returns the value of the damping-constant parameter for DC motors. Sets the parameter if v is given.

"dc_integration_constant" – returns the value of the integration-constant parameter for DC motors. Sets the parameter if v is given.

"dc_integration_limit" – returns the value of the integration-limit parameter for DC motors. Sets the parameter if v is given.

"dc_following_error" – returns the value of the dc-following parameter for DC motors. Sets the parameter if v is given.

"dc_sampling_interval" – returns the value of the sampling-interval parameter for DC motors. Sets the parameter if v is given.

"encoder_step_size" – returns the value of the encoder step size parameter. Sets the parameter if v is given.

"step_mode" – returns the value of the step-mode parameter. Sets the parameter if v is given. A zero indicates full-step mode, while a one indicates half-step mode.

"deceleration" – returns the value of the deceleration parameter. Sets the parameter if v is given.

"torque" – returns the value of the torque parameter. Sets the parameter if v is given.

Rereading the *config* file resets the values of all the motor parameters to the values in the *config* file. Only moderate consistency checking is done by `spec` on the values programmed with `motor_par()`. Be sure to use values meaningful to your particular motor controller.

`get_lim(i , w)` – Returns the dial limit of motor i . If $w > 0$, returns high limit. If $w < 0$, returns low limit.

`set_lim(i , u , v)` – Sets the low and high dial limits of motor i . It doesn't matter which order the limits, u and v , are given. Returns -1 if not configured for motor i or if the motor is protected, unusable or moving, else returns 0 .

`dial(i , u)` – Returns the motor dial position for motor i corresponding to user angle u .

`user(i , d)` – Returns the user angle for motor i corresponding to dial position u .

`chg_dial(i , u)` – Sets the dial position of motor i to u by changing the contents of the controller registers. Returns -1 if not configured for motor i or if the motor is protected,

unusable or moving, else returns 0.

`chg_dial(i, s [, u])` – Starts motor *i* on a home or limit search, according to the value of *s*, as follows:

"home+" – move to home switch in positive direction.

"home-" – move to home switch in negative direction.

"home" – move to home switch in positive direction if current dial position is less than zero, otherwise move to home switch in negative direction.

"lim+" – move to limit switch in positive direction.

"lim-" – move to limit switch in negative direction.

Positive and negative direction are with respect to the dial position of the motor. (Not all motor controllers implement the home or limit search feature.) If present, the value of the third argument is used to set the motor's dial position when the home or limit position is reached (as of `spec` release 4.05.10-3). Returns -1 if not configured for motor *i* or if the motor is protected, unusable or moving, else returns 0.

`chg_offset(i, u)` – Sets offset (determining user angle) of motor *i* to *u*. Returns -1 if not configured for motor *i* or if the motor is unusable or moving, else returns 0.

MOTOR NUMBERING

For most motor controllers, unit and channel numbers can be explicitly assigned to individual motors channels in the hardware configuration editor. The unit numbers start from zero for each different model of motor controller and are implicitly assigned to consecutive units as they are listed in the configuration editor.

Channel numbers start from zero for most motor controllers. For the following controllers, where the hardware clearly labels the motor channels, the channel numbers start from one: AllMotion EZStepper, Attocube ANC350, Bruker D8, Compumotor 6K, Delta Tau PMAC/PMAC2, EPICS standard motor record, HANARO MCU, HMT HCC1, IMS MDrive, IMS MicroLYNX 4/7, JVL SMI20B, Kohzu Seiki PMC-2/3/4 GT, Kohzu SC-200/400/800, Labo NT-2400, MAC Science MXC, Micos VENUS-2/3 Compatible, MicroMo MVP-2001, Munich BR-tronik IPS, Newport Agilis, Physik Instrumente C-630, Physik Instrumente E-710 and E-712, PMC DCX-100 and DCX-PCI300, Rigaku RINT-2000, RISO ECB, SCIPE, Siemens D5000, Sigmatech FC-501A and Velmex VXM-1/VXM-2.

If the optional unit/channel numbering is left blank, motors are assigned to motor controller channels in the order they each appear in the `config` file. For example, if several motor controllers of the same type are configured and each can control eight motors, the first eight motors configured are assigned to the first controller, the next eight, to the next controller, and so on. If a controller channel is unusable, a motor must still be configured for that channel, although if given the name "unused", information about that motor won't be displayed by the standard macros.

Note, support for the following older motor controllers hasn't been updated to recognize unit/channel numbering: Joerger SMC, Oriel 18011, Compumotor 3000/4000 and SX, Micro-Controle IP28 and SIX19, Klinger MC4, ACS MCB, NSLS MMC32, NSK and Inel XRGCI. For these controllers, the consecutive numbering rule applies.

Standard macros such as `wh` display only the motors most relevant to a particular geometry. Other macros that display information about all the motors, such as `wa`, list these most relevant motors first. For a particular geometry configuration, the relevant motors are established in the `_assign` macro, which is normally in geometry-specific macro file. A user can change the order in which the motors are displayed by redefining the `_assign` macro.

MOTION

Motors are set in motion by the command `move_all`, which simultaneously moves all motors to the positions in the array `A[]` (in user units). Note, though, the standard macros use `move_em`, which incorporates the `user_premove` and `user_postmove` user-hook macros before and after the `move_all` call. It is recommended that local macros also use `move_em` rather than `move_all`.

When `move_all` is executed, the code makes several checks on the validity of the move, including checks on whether any of the dial positions corresponding to the user angles in `A[]` are outside the dial limits. If so, no motors are moved, and the program responds with an error.

Note, moving motors is asynchronous. The program returns while the motors are still in motion. However, moving motors can be synchronized using the function `wait()`.

When writing macros to move motors, you should always wait for motors to stop, do a `get_angles` command to fill `A[]` with the current positions, assign new values to the elements of `A[]` for the motors that are to be moved, and then do the `move_em`.

Study the definitions of the standard macros `mv`, `umv`, `br`, `an`, `pl`, etc., to see normal usage.

SECURE MOTORS

At some spec installations, the spec administrator may prevent users from either moving particular motors or changing the limits of particular motors.

EXAMPLE

Here is a primitive macro to move one motor:

```
def mv '
  if ($# != 2) {
    print "usage:  mv motor position"
    exit
  }
  wait()          # Let previous motions finish.
  get_angles     # Make sure A[] is current so no
                # other motors start moving.
  A[$1]=$2      # Change the relevant A[].
  move_em       # Start the motors.
,
```

SEE ALSO

`config_adm` `lm` `mv` `mvd` `mvr` `set` `set_dial` `set_lm` `tw` `w` `wa` `wm`